BACKEND DOCUMENTATION: APPOINTMENT ORGANISER

1. Introduction

The backend repository for the **Appointment Organizer** project is responsible for managing business logic, data storage, and API endpoints. It serves as the backbone of the application, facilitating user authentication, event organization, and database interactions.

This documentation serves as a complete guide to understanding and working with the backend repository of the Appointment Organizer project.

Technologies Used

- **Node.js**: Backend runtime environment.
- **Express.js**: Web framework for API development.
- **Prisma**: ORM for interacting with the PostgreSQL database.
- **PostgreSQL**: Relational database system.
- **Swagger**: API documentation and testing.

2. Project Setup

Prerequisites

- **Node.js**
- **PostgreSQL**
- Install the Prisma CLI globally: npm install -g prisma

Installation

1. Clone the repository: git clone https://github.com/HSFD-WS-24-25/backend.git
2. Navigate to the project directory: cd backend-repo
3. Install dependencies: npm install

Environment Configuration

1. Create a .env file in the root directory by copying .env.example: cp .env.example .env
2. Fill in the required environment variables:
   DATABASE_URL=postgresql://<username>:<password>@localhost:5432/<database_name>

3. PORT=3000

3. Folder Structure

Core Folders of the repository

1. **.github/workflows**: that contains CI/CD pipeline configuration (main_eventplanner-backend.yml).

1. **config**:
    - constants.js: to stores reusable constants.
    - messages.js: to centralized response messages.
    - permissions.js, roles.js, statusCodes.js: to define role-based access control and HTTP status codes.
    - **database/prisma.js**: Initializes and exports the Prisma client.
2. **controllers**: for the business logic for handling requests:
        1. eventControllers.js: to handles event-related endpoints.
        2. organizationController.js: to manages organisations.
        3. userController.js: to manages users and authentication.
3. **helpers**: authHelper.js: use to provides utility functions for authentication (example for token generation).
4. **middleware**:
    - authMiddleware.js: to handles authentication checks.
    - permissionMiddleware.js: to verifies user permissions.
    - validationMiddleware.js: to validates request payloads.
    - logTokenMiddleware.js: to logs and monitors token activity.
5. **models**:
    - users.json: which is an example data or user model.
6. **prisma**:
    - schema.prisma: use to defines the database schema using Prisma syntax.
        1. **Migrations**: use to stores SQL migration files (example: add_all_database_tables/migration.sql.)
7. **routes/api**: which defines API endpoints:

1. events.js: Event-related routes.
2. organisations.js: Organization-related routes.
3. users.js: User-related routes.
4. index.js: Combines all route files.

8. **seeders**: that contains the database seeding scripts such as nukeDB.js, runSeeds.js, seedEvents.js, seedOrganisations.js, seedRoleAndPermissions.js, seedUsers.js etc.
9. **services**: that contains emailServices.js which handles email-related logic (example: sending notifications).
10. **swagger**: that contains swagger.js and swaggerDocs.js which is use to configure Swagger for API documentation.
11. **validators**: contains eventValidator.js which is use to validate event-related payloads.

4. API Documentation

Base URL

- Local development: http://localhost:3000

* USERS

1 GET /api/users:

Description: Retrieve all users in the system.

Response example:

```
[
  {
    "id": "0123",
    "name": "Max Mus",
    "email": "max.mus @example.com",
    "role": "organizer"
  }
]
```

2 POST /api/users:

Description: Create a new user.

Request Body:

```
{
  "name": "Max Mus",
  "email": "max.mus@example.com",
  "password": "password",
  "role": "participant"
```

```
}
```

Response example:

```
{
 "id": "0123",
 "name": "Max Mus",
 "email": "max.mus@example.com",
 "role": "participant"
}
```

3 GET /api/users/:id:

Description: Retrieve user details by their ID.

Response example:

```
{
 "id": "0123",
 "name": "Max Mus",
 "email": "max.mus@example.com",
 "role": "participant"
}
```

*** EVENTS**

1. GET /api/events:

Description: Retrieve all events.

Response example:

```
[
  {
   "id": "456",
   "title": "Goebel Meeting",
   "description": "Weekly Goebel meeting",
   "date": "2025-02-04",
   "organizer": {
    "id": "123",
    "name": "Max Mus"
   }
  }
]
```

2. POST /api/events:

Description: Create a new event.

Request Body:

```
{
 "title": "Bachelor Project",
 "description": "Discussion about the current bachelor project",
```

```
  "date": "2025-02-11",
  "organizerId": "123"
}
```

Response example:

```
{
  "id": "456",
  "title": " Bachelor Project",
  "description": "Discussion about the current bachelor project ",
  "date": "2025-01-11",
  "organizer": {
    "id": "123",
    "name": "Max Mus "
  }
}
```

*** ORGANISATIONS**

1. GET /api/organisations:

   Description: List all organisations.

   Response example:

```
[
  {
    "id": "789",
    "name": "Backend Team",
    "description": "Best Team for the year 2025"
  }
]
```

2. POST /api/organisations:

   Description: Create a new organization.

   Request Body:

```
{
  "name": "Backend Team",
  "description": " Best Team for the year 2025"
}
```

   Response Example:

```
{
  "id": "789",
  "name": "Backend Team",
  "description": " Best Team for the year 2025"
}
```

5. Database (example of prisma schema):

```
example User {
  id            String  @id @default(uuid())
  name          String
  email         String  @unique
  role          String
  password      String
  events        Event[]
  createdAt     DateTime @default(now())
  updatedAt     DateTime @updatedAt
}


example Event {
  id            String  @id @default(uuid())
  title         String
  description   String
  date          DateTime
  organizerId   String
  organizer     User    @relation(fields: [organizerId], references: [id])
  createdAt     DateTime @default(now())
  updatedAt     DateTime @updatedAt
}


example Organization {
  id            String  @id @default(uuid())
  name          String
  description   String
  users         User[]
  createdAt     DateTime @default(now())
  updatedAt     DateTime @updatedAt
}
```

Run migrations with: npx prisma migrate dev

6. Middleware handle tasks such as:

- Authenticating requests (authMiddleware.js).
- Validating request payloads (validationMiddleware.js).
- Checking user permissions (permissionMiddleware.js).

7. Deployment

Local Deployment:

1. Start the server: npm start
2. Swagger documentation is available at: http://localhost:3000/swagger/swaggerDocs.js

Production Deployment:

- Deploy the application to a cloud platform (example Azure etc.).

8. Contribution Guidelines

Guidelines

- **Branching**: Use feature branches for new additions.
- **Testing**: Add unit tests for new features using Jest or a similar framework.